Linear Time Feature Selection for Regularized Least-Squares

Tapio Pahikkala, Antti Airola, and Tapio Salakoski

March 19, 2010

Abstract

We propose a novel algorithm for greedy forward feature selection for regularized least-squares (RLS) regression and classification, also known as the least-squares support vector machine or ridge regression. The algorithm, which we call greedy RLS, starts from the empty feature set, and on each iteration adds the feature whose addition provides the best leave-one-out cross-validation performance. Our method is considerably faster than the previously proposed ones, since its time complexity is linear in the number of training examples, the number of features in the original data set, and the desired size of the set of selected features. Therefore, as a side effect we obtain a new training algorithm for learning sparse linear RLS predictors which can be used for large scale learning. This speed is possible due to matrix calculus based short-cuts for leave-one-out and feature addition. We experimentally demonstrate the scalability of our algorithm and its ability to find good quality feature sets.

1 Introduction

Feature selection is one of the fundamental tasks in machine learning. Simply put, given a set of features, the task is to select an informative subset. The selected set can be informative in the sense that it guarantees a good performance of the machine learning method or that it will provide new knowledge about the task in question. Such increased understanding of the data is sought after especially in life sciences, where feature selection is often used, for example, to find genes relevant to the problem under consideration. Other benefits include compressed representations of learned predictors and faster prediction time. This can enable the application of learned models when constrained by limited memory and real-time response demands, as is typically the case in embedded computing, for example.

The feature selection methods are divided by [1] into three classes, the so called filter, wrapper, and embedded methods. In the filter model feature selection is done as a preprocessing step by measuring only the intrinsic properties of the data. In the wrapper model [2] the features are selected through interaction with a machine learning method. The quality of selected features is evaluated by measuring some estimate of the generalization performance of a prediction model constructed using them. The embedded methods minimize an objective function which is a trade-off between a goodness-of-fit term and a penalty term for a large number of features (see e.g. [3]).

On one hand, the method we propose in this work is a wrapper method, since equivalent results can be obtained by using a black-box learning algorithm together with the considered search and performance estimation strategies. On the other hand it may also be considered as an embedded method, because the feature subset selection is so deeply integrated in the learning process, that our work results in a novel efficient training algorithm for the method.

As suggested by [4], we estimate the generalization performance of models learned with different subsets of features with cross-validation. More specifically, we consider the leave-one-out cross-validation (LOO) approach, where each example in turn is left out of the training set and used for testing. Since the num-

ber of possible feature combinations grows exponentially with the number of features, a search strategy over the power set of features is needed. The strategy we choose is the greedy forward selection approach. The method starts from the empty feature set, and on each iteration adds the feature whose addition provides the best leave-one-out cross-validation performance. That is, it performs a steepest descent hill-climbing in the space of feature subsets of size up to a given limit $k \in \mathbb{N}$ and is greedy in the sense that it does not remove features once they have been selected.

Our method is built upon the Regularized least-squares (RLS), also known as the least-squares support vector machine (LS-SVM) and ridge regression, which is is a state-of-the art machine learning method suitable both for regression and classification [5–11]. An important property of the algorithm is that it has a closed form solution, which can be fully expressed in terms of matrix operations. This allows developing efficient computational shortcuts for the method, since small changes in the training data matrix correspond to low-rank changes in the learned predictor. Especially it makes possible the development of efficient cross-validation algorithms. An updated predictor, corresponding to a one learned from a training set from which one example has been removed, can be obtained via a well-known computationally efficient short-cut (see e.g. [12–14]) which in turn enables the fast computation of LOO-based performance estimates. Analogously to removing the effect of training examples from learned RLS predictors, the effects of a feature can be added or removed by updating the learned RLS predictors via similar computational short-cuts.

Learning a linear RLS predictor with k features and m training examples requires $O(\min\{k^2m, km^2\})$ time, since the training can be performed either in primal or dual form depending whether $m > k$ or vice versa. Given that the computation of LOO performance requires m retrainings, that the forward selection goes through of the order of $O(n)$ features available for selection in each iteration, and that the forward selection has k iterations, the overall time complexity of the forward selection with LOO criterion becomes $O(\min\{k^3m^2n, k^2m^3n\})$ in case RLS is used as a black-box method.

In machine learning and statistics literature, there have been several studies in which the computational short-cuts for LOO, as well as other types of cross-validation, have been used to speed up the evaluation of feature subset quality for ordinary non-regularized least-squares (see e.g [15, 16]) and for RLS (see e.g. [17,18]). However, the considered approaches are still be computationally quite demanding, since the LOO estimate needs to be re-calculated from scratch for each considered subset of features. Recently, [19] introduced a method which uses additional computational short-cuts for efficient updating of the LOO predictions when adding new features to those already selected. The computational cost of the incremental forward selection procedure is only $O(km^2n)$ for the method. The speed improvement is notable especially in cases, where the aim is to select a large number of features but the training set is small. However, the method is still impractical for large training sets due to its quadratic scaling.

In this paper, we improve upon the above results by showing how greedy forward selection according to the LOO criterion can be carried out in $O(kmn)$ time. Thus, our algorithm, which we call greedy RLS or greedy LS-SVM, is linear in the number of examples, features, and selected features, while it still provides the same solution as the straightforward wrapper approach. As a side effect we obtain a novel training algorithm for linear RLS predictors that is suitable for large-scale learning and guarantees sparse solutions. Computational experiments demonstrate the scalability and efficiency of greedy RLS, and the predictive power of selected feature sets.

2 Regularized Least-Squares

We start by introducing some notation. Let $\mathbb{R}^m$ and $\mathbb{R}^{n\times m}$, where $n, m \in \mathbb{N}$, denote the sets of real valued column vectors and $n \times m$-matrices, respectively. To denote real valued matrices and vectors we use capital letters and bold lower case letters, respectively. Moreover, index sets are denoted with calligraphic capital letters. By denoting M_i, $M_{:,j}$, and $M_{i,j}$, we refer to the ith row, jth column, and i, jth entry of

the matrix $M \in \mathbb{R}^{n \times m}$, respectively. Similarly, for index sets $\mathcal{R} \in \{1, \ldots, n\}$ and $\mathcal{L} \in \{1, \ldots, m\}$, we denote the submatrices of M having their rows indexed by $\mathcal{R}$, the columns by $\mathcal{L}$, and the rows by $\mathcal{R}$ and columns by $\mathcal{L}$ as $M_{\mathcal{R}}$, $M_{:,\mathcal{L}}$, and $M_{\mathcal{R},\mathcal{L}}$, respectively. We use an analogous notation also for column vectors, that is, $\mathbf{v}_i$ refers to the ith entry of the vector $\mathbf{v}$.

Let $X \in \mathbb{R}^{n \times m}$ be a matrix containing the whole feature representation of the examples in the training set, where n is the total number of features and m is the number of training examples. The i, jth entry of X contains the value of the ith feature in the jth training example. Moreover, let $\mathbf{y} \in \mathbb{R}^m$ be a vector containing the labels of the training examples. In binary classification, the labels can be restricted to be either -1 or 1, for example, while they can be any real numbers in regression tasks.

In this paper, we consider linear predictors of type

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \mathbf{x}_{\mathcal{S}}, \tag{1}$$

where $\mathbf{w}$ is the $|\mathcal{S}|$-dimensional vector representation of the learned predictor and $\mathbf{x}_{\mathcal{S}}$ can be considered as a mapping of the data point x into $|\mathcal{S}|$-dimensional feature space.[1] Note that the vector $\mathbf{w}$ only contains entries corresponding to the features indexed by $\mathcal{S}$. The rest of the features of the data points are not used in prediction phase. The computational complexity of making predictions with (1) and the space complexity of the predictor are both $O(k)$, where k is the desired number of features to be selected, provided that the feature vector representation $\mathbf{x}_{\mathcal{S}}$ for the data point x is given.

Given training data and a set of feature indices $\mathcal{S}$, we find $\mathbf{w}$ by minimizing the so-called regularized least-squares (RLS) risk. Minimizing the RLS risk can be expressed as the following problem:

$$\underset{\mathbf{w} \in \mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}} ((\mathbf{w}^{\mathrm{T}} X_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y})^{\mathrm{T}} ((\mathbf{w}^{\mathrm{T}} X_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y}) + \lambda \mathbf{w}^{\mathrm{T}} \mathbf{w}. \tag{2}$$

The first term in (2), called the empirical risk, measures how well the prediction function fits to the training data. The second term is called the regularizer and it controls the tradeoff between the loss on the training set and the complexity of the prediction function.

A straightforward approach to solve (2) is to set the derivative of the objective function with respect to $\mathbf{w}$ to zero. Then, by solving it with respect to $\mathbf{w}$, we get

$$\mathbf{w} = (X_{\mathcal{S}}(X_{\mathcal{S}})^{\mathrm{T}} + \lambda I)^{-1} X_{\mathcal{S}} \mathbf{y}, \tag{3}$$

where $I \in \mathbb{R}^{m \times m}$ is the identity matrix. We note (see e.g. [20]) that an equivalent result can be obtained from

$$\mathbf{w} = X_{\mathcal{S}}((X_{\mathcal{S}})^{\mathrm{T}} X_{\mathcal{S}} + \lambda I)^{-1} \mathbf{y}. \tag{4}$$

If the size of the set $\mathcal{S}$ of currently selected features is smaller than the number of training examples m, it is computationally beneficial to use the form (3) while using (4) is faster in the opposite case. Namely, the computational complexity of the former is $O(|\mathcal{S}|^3 + |\mathcal{S}|^2 m)$, while the that of the latter is $O(m^3 + m^2 |\mathcal{S}|)$, and hence the complexity of training a predictor is $O(\min\{|\mathcal{S}|^2 m, m^2 |\mathcal{S}|\})$.

To support the considerations in the following sections, we introduce the dual form of the prediction function and some extra notation. According to [7], the prediction function (1) can be represented in dual form as follows

$$f(\mathbf{x}) = \mathbf{a}^{\mathrm{T}} (X_{\mathcal{S}})^{\mathrm{T}} \mathbf{x}_{\mathcal{S}}.$$

Here $\mathbf{a} \in \mathbb{R}^m$ is the vector of so-called dual variables, which can be obtained from

$$\mathbf{a} = G\mathbf{y},$$

where

$$G = (K + \lambda I)^{-1} \tag{5}$$

and

$$K = (X_{\mathcal{S}})^{\mathrm{T}} X_{\mathcal{S}}. \tag{6}$$

Note that if $\mathcal{S} = \emptyset$, K is defined to be a matrix whose every entry is equal to zero.

In the context of kernel-based learning algorithms (see e.g. [21–23]), K and $\mathbf{a}$ are usually called the kernel matrix and the vector of dual variables, respectively. The kernel matrix contains the inner products between the feature vectors of all training data

[1] In the literature, the formula of the linear predictors often also contain a bias term. Here, we assume that if such a bias is used, it will be realized by using an extra constant valued feature in the data points.

points. With kernel methods, the feature vector representation of the data points may be even infinite dimensional as long as there is an efficient, although possibly implicit, way to calculate the value of the inner product and the dual variables are used to represent the prediction function. However, while the dual representation plays an important role in the considerations of this paper, we restrict our considerations to feature representations of type $\mathbf{x}_{\mathcal{S}}$ which can be represented explicitly.

Now let us consider some well-known efficient approaches for evaluating the LOO performance of a trained RLS predictor (see e.g. [12–14, 24] and for the non-regularized case, see e.g [15, 16]). The LOO prediction for the jth training example can be obtained in constant time from

$$(1 - \mathbf{q}_j)^{-1}(\mathbf{f}_j - \mathbf{q}_j \mathbf{y}_j), \tag{7}$$

where

$$\mathbf{f} = (\mathbf{w}^{\mathrm{T}} X_{\mathcal{S}})^{\mathrm{T}}$$

and

$$\mathbf{q}_j = (X_{\mathcal{S},j})^{\mathrm{T}} (X_{\mathcal{S}} (X_{\mathcal{S}})^{\mathrm{T}} + \lambda I)^{-1} X_{\mathcal{S},j},$$

provided that the vectors $\mathbf{f}$ and $\mathbf{q}$ are computed and stored in memory. The time complexity of computing $\mathbf{f}$ and $\mathbf{q}$ is $O(|\mathcal{S}|^3 + |\mathcal{S}|^2 m)$, which is the same as that of training RLS via (3). Alternatively, the constant time LOO predictions can be expressed in terms of the dual variables $\mathbf{a}$ and the diagonal entries of the matrix G as follows:

$$\mathbf{y}_j - (G_{j,j})^{-1} \mathbf{a}_j. \tag{8}$$

Here, the time complexity of computing $\mathbf{a}$ and the diagonal entries of the matrix G is $O(m^3 + m^2|\mathcal{S}|)$, which is equal to the complexity of training RLS via (4). Thus, using either (7) or (8), the LOO performance for the whole training set can be computed in $O(\min\{|\mathcal{S}|^2 m, m^2|\mathcal{S}|\})$ time, which is equal to the time complexity of training RLS.

3 Feature Selection

In this section, we consider algorithmic implementations of feature selection for RLS with leave-one-out (LOO) criterion. We start by considering a straightforward wrapper approach which uses RLS as a black-box method in Section 3.1. Next, we recall a previously proposed algorithm which provides results that are equivalent to those of the wrapper approach but which has computational short-cuts to speed up the feature selection in Section 3.2. In Section 3.3, we present greedy RLS, our novel linear time algorithm, which again provides the same results as the wrapper approach but which has much lower computational and memory complexity than the previously proposed algorithms.

3.1 Wrapper Approach

Algorithm 1: Standard wrapper algorithm for RLS

Input: $X \in \mathbb{R}^{n \times m}$, $\mathbf{y} \in \mathbb{R}^m$, k, λ
Output: $\mathcal{S}$, $\mathbf{w}$

```
S ← ∅;
while |S| < k do
    e ← ∞;
    b ← 0;
    foreach i ∈ {1, ..., n} \ S do
        R ← S ∪ {i};
        e_i ← 0;
        foreach j ∈ {1, ..., m} do
            L ← {1, ..., m} \ {j};
            w ← t(X_{RL}, y_L, λ);
            p ← w^T X_{R,j};
            e_i ← e_i + l(y_j, p);
        end
        if e_i < e then
            e ← e_i;
            b ← i;
        end
    end
    S ← S ∪ {b};
end
w ← t(X_S, y, λ);
```

Here, we consider the standard wrapper approach in which RLS is used as a black-box method which is re-trained for every feature set to be evaluated during

the selection process and for every round of the LOO cross-validation. In forward selection, the set of all feature sets up to size k is searched using hill climbing in the steepest descent direction (see e.g. [25]) starting from an empty set. The method is greedy in the sense that it adds one feature at a time to the set of selected features but no features are removed from the set at any stage. The wrapper approach is presented in Algorithm 1. In the algorithm description, $t(\cdot,\cdot,\cdot)$ denotes the black-box training procedure for RLS which takes a data matrix, a label vector, and a value of the regularization parameter as input and returns a vector representation of the learned predictor $\mathbf{w}$. The function $l(\cdot,\cdot)$ computes the loss for one training example given its true label and a predicted label obtained from the LOO procedure. Typical losses are, for example, the squared loss for regression and zero-one error for classification.

Given that the computation of LOO performance requires m retrainings, that the forward selection goes through $O(n)$ features in each iteration, and that k features are chosen, the overall time complexity of the forward selection with LOO criterion is $O(\min\{k^3m^2n, k^2m^3n\})$. Thus, the wrapper approach is feasible with small training sets and in cases where the aim is to select only a small number of features. However, at the very least quadratic complexity with respect to both the size of the training set and the number of selected features make the standard wrapper approach impractical in large scale learning settings. The space complexity of the wrapper approach is $O(nm)$ which is equal to the cost of storing the data matrix X.

An immediate reduction for the above considered computational complexities can be achieved via the short-cut methods for calculating the LOO performance presented in Section 2. In machine learning and statistics literature, there have been several studies in which the computational short-cuts for LOO, as well as other types of cross-validation, have been used to speed up the evaluation of feature subset quality for non-regularized least-squares (see e.g [15,16]) and for RLS (see e.g. [17, 18]). For the greedy forward selection, the short-cuts reduce the overall computational complexity to $O(\min\{k^3mn, k^2m^2n\})$, since LOO can then be calculated as efficiently as training the predictor itself. However, both [18] and [17] considered only the dual formulation (8) of the LOO speed-up which is expensive for large data sets. Thus, we are not aware of any studies in which the primal formulation (7) would be implemented for the greedy forward selection for RLS.

3.2 Previously Proposed Speed-Up

We now present a feature selection algorithm proposed by [19] which the authors call low-rank updated LS-SVM. The features selected by the algorithm are equal to those by standard wrapper algorithm and by the method proposed by [17] but the method has a lower time complexity with respect to k due to certain matrix calculus based computational short-cuts. The low-rank updated LS-SVM algorithm for feature selection is presented in Algorithm 2.

The low-rank updated LS-SVM is based on updating the matrix G and the vector $\mathbf{a}$ as new features are added into $\mathcal{S}$. In the beginning of the forward update algorithm, the set $\mathcal{S}$ of selected features is empty. This means that the matrix G contains no information of the data matrix X. Instead, every entry of the matrix K is equal to zero, and hence G and $\mathbf{a}$ are initialized to $\lambda^{-1}I$ and $\lambda^{-1}\mathbf{y}$, respectively, in lines 2 and 3 in Algorithm 2. When a new feature candidate is evaluated, the LOO performance corresponding to the updated feature set can be calculated using the temporarily updated G and $\mathbf{a}$, denoted in the algorithm as $\widetilde{G}$ and $\tilde{\mathbf{a}}$, respectively. Here, the improved speed is based on two short-cuts. Namely, updating the matrix G via the well known Sherman-Morrison-Woodbury (SMW) formula and using the fast LOO computation.

Let us first consider updating the matrix G. Formally, if the ith feature is to be evaluated, where $i \notin \mathcal{S}$, the algorithm calculates the matrix $\widetilde{G}$ corresponding the feature index set $\mathcal{S} + \{i\}$. According to (5), the matrix $\widetilde{G}$ is

$$(K + \mathbf{v}\mathbf{v}^{\mathrm{T}} + \lambda I)^{-1}, \tag{9}$$

where K is the kernel matrix defined as in (6) without having the ith feature in $\mathcal{S}$ and $\mathbf{v} = (X_i)^{\mathrm{T}}$ contains the values of the ith feature for the training data.

Algorithm 2: Low-rank updated LS-SVM

Input: $X \in \mathbb{R}^{n \times m}$, $\mathbf{y} \in \mathbb{R}^m$, k, λ
Output: $\mathcal{S}$, $\mathbf{w}$

1 $\mathcal{S} \leftarrow \emptyset$;
2 $\mathbf{a} \leftarrow \lambda^{-1}\mathbf{y}$;
3 $G \leftarrow \lambda^{-1}I$;
4 **while** $|\mathcal{S}| < k$ **do**
5 $e \leftarrow \infty$;
6 $b \leftarrow 0$;
7 **foreach** $i \in \{1, \ldots, n\} \setminus \mathcal{S}$ **do**
8 $\mathbf{v} \leftarrow (X_i)^{\mathrm{T}}$;
9 $\widetilde{G} \leftarrow G - G\mathbf{v}(1 + \mathbf{v}^{\mathrm{T}}G\mathbf{v})^{-1}(\mathbf{v}^{\mathrm{T}}G)$;
10 $\tilde{\mathbf{a}} \leftarrow \widetilde{G}\mathbf{y}$;
11 $e_i \leftarrow 0$;
12 **foreach** $j \in \{1, \ldots, m\}$ **do**
13 $p \leftarrow \mathbf{y}_j - (\widetilde{G}_{j,j})^{-1}\tilde{\mathbf{a}}_j$;
14 $e_i \leftarrow e_i + l(\mathbf{y}_j, p)$;
15 **end**
16 **if** $e_i < e$ **then**
17 $e \leftarrow e_i$;
18 $b \leftarrow i$;
19 **end**
20 **end**
21 $\mathbf{v} \leftarrow (X_b)^{\mathrm{T}}$;
22 $G \leftarrow G - G\mathbf{v}(1 + \mathbf{v}^{\mathrm{T}}G\mathbf{v})^{-1}(\mathbf{v}^{\mathrm{T}}G)$;
23 $\mathbf{a} \leftarrow G\mathbf{y}$;
24 $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
25 **end**
26 $\mathbf{w} \leftarrow X_{\mathcal{S}}\mathbf{a}$;

The time needed to compute (9) is cubic in m, which provides no benefit compared to the standard wrapper approach. However, provided that the matrix G is stored in memory, it is possible to speed up the computation of $\widetilde{G}$ via the SMW formula. Formally, the matrix $\widetilde{G}$ can also be obtained from

$$G - G\mathbf{v}(1 + \mathbf{v}^{\mathrm{T}}G\mathbf{v})^{-1}(\mathbf{v}^{\mathrm{T}}G) \tag{10}$$

which requires a time quadratic in m provided that the multiplications are performed in the order indicated by the parentheses. Having calculated $\widetilde{G}$, the updated vector of dual variables $\tilde{\mathbf{a}}$ can be obtained from

$$\widetilde{G}\mathbf{y} \tag{11}$$

also with $O(m^2)$ floating point operations.

Now let us consider the efficient evaluation of the features with LOO performance. Provided that $\widetilde{G}$ and $\tilde{\mathbf{a}}$ are already stored in memory, the LOO performance of RLS trained with the $\mathcal{S} + \{i\}$ training examples can be calculated via (8). After the feature, whose inclusion would be the most favorable with respect to the LOO performance, is found, it is added to the set of selected features and RLS solution is updated accordingly in the last four lines of the algorithm.

The outermost loop is run k times, k denoting the number of features to be selected. For each round of the outer loop, the middle loop is run $O(n)$ times, because each of the $n - |\mathcal{S}|$ features is evaluated in turn before the best of them is added to the set of selected features $\mathcal{S}$. Due to the efficient formula (8) for LOO computation, the calculation of the LOO predictions for the m requires only $O(m)$ floating point operations in each run of the innermost loop. The complexity of the middle loop is dominated by the $O(m^2)$ time needed for calculating $\widetilde{G}$ in line 9 and $\tilde{\mathbf{a}}$ in line 10, and hence the overall time complexity of the low-rank updated LS-SVM algorithm is $O(knm^2)$.

The complexity with respect to k is only linear which is much better than that of the standard wrapper approach, and hence the selection of large feature sets is made possible. However, feature selection with large training sets is still infeasible because of the quadratic complexity with respect to m. In fact, the running time of the standard wrapper approach with

the LOO short-cut can be shorter than that of the low-rank updated LS-SVM method in cases where the training set is large and the number of features to be selected is small.

The space complexity of the low-rank updated LS-SVM algorithm is $O(nm+m^2)$, because the matrices X and G require $O(nm)$ and $O(m^2)$ space, respectively. Due to the quadratic dependence of m, this space complexity is worse than that of the standard wrapper approach.

3.3 Linear Time Forward Selection Algorithm

Here, we present our novel algorithm for greedy forward selection for RLS with LOO criterion. We refer to our algorithm as greedy RLS, since in addition to feature selection point of view, it can also be considered as a greedy algorithm for learning sparse RLS predictors. Our method resembles the low-rank updated LS-SVM in the sense that it also operates by iteratively updating the vector of dual variables, and hence we define it on the grounds of the considerations of Section 3.2. Pseudo code of greedy RLS is presented in Algorithm 3.

The time and space complexities of the low-rank updated LS-SVM algorithm are quadratic in m, because it involves updating the matrix G (lines 9 and 22 in Algorithm 2) and the vector $\mathbf{a}$ (lines 10 and 23 in Algorithm 2). In order to improve the efficiency of the algorithm by getting rid of the quadratic dependence of m in both space and time, we must avoid the explicit calculation and storing of the matrices G and $\widetilde{G}$. Next, we present an improvement which solves these problems.

In the middle loop of the low-rank updated LS-SVM algorithm in Algorithm 2, it is assumed that the matrix G constructed according to the current set $\mathcal{S}$ of selected features is stored in memory. It is temporarily updated to matrix $\widetilde{G}$ corresponding to the set $\mathcal{S}+\{i\}$, where i is the index of the feature to be evaluated. We observe that the LOO computations via the formula (8) require the vector of dual variables $\tilde{\mathbf{a}}$ corresponding to $\mathcal{S}+\{i\}$ but only the diagonal elements of $\widetilde{G}$.

Let us first consider speeding up the computation of $\tilde{\mathbf{a}}$. Since we already have $\mathbf{a}$ stored in memory, we can take advantage of it when computing $\tilde{\mathbf{a}}$. That is, since the value of the vector $\mathbf{a}$ before the update is $G\mathbf{y}$, its updated value $\tilde{\mathbf{a}}$ can be, according to (10) and (11), obtained from

$$\mathbf{a} - G\mathbf{v}(1+\mathbf{v}^{\mathrm{T}}G\mathbf{v})^{-1}(\mathbf{v}^{\mathrm{T}}\mathbf{a}), \tag{12}$$

where $\mathbf{v} = (X_i)^{\mathrm{T}}$. This does not yet help us much, because the form (12) still contains the matrix G explicitly. However, if we assume that, in addition to $\mathbf{a}$, the product $G\mathbf{v}$ is calculated in advance and the result is stored in memory, calculating (12) only requires $O(m)$ time. In fact, since the operation (12) is performed for almost every feature during each round of the greedy forward selection, we assume that we have an $m \times n$ cache matrix, denoted by

$$C = GX^{\mathrm{T}}, \tag{13}$$

computed in advance containing the required products for all features, each column corresponding to one product.

Now, given that C is available in memory, we calculate, in $O(m)$ time, a vector

$$\mathbf{u} = C_{:,i}(1+\mathbf{v}^{\mathrm{T}}C_{:,i})^{-1} \tag{14}$$

and store it in memory for later usage. This is done in lines 10 and 24 of the algorithm in Algorithm 3. Consequently, (12) can be rewritten as

$$\mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a}). \tag{15}$$

and hence the vector $\mathbf{a}$ can be updated in $O(m)$ time.

We also have to ensure that fast computation of the LOO performance. The use of (8) for computing the LOO for the updated feature set requires the diagonal elements of the matrix $\tilde{G}$. Let $\mathbf{d}$ and $\tilde{\mathbf{d}}$ denote the vectors containing the diagonal elements of G and $\tilde{G}$, respectively. We observe that the SMW formula (10) for calculating $\tilde{G}$ can be rewritten as

$$G - \mathbf{u}(C_{:,i})^{\mathrm{T}}. \tag{16}$$

Now, we make an assumption that, instead of having the whole G being stored in memory, we have only

stored its diagonal elements, that is, the vector $\mathbf{d}$. Then, according to (16), the jth element of the vector $\tilde{\mathbf{d}}$ can be computed from

$$\mathbf{d}_j - \mathbf{u}_j C_{j,i} \tag{17}$$

requiring only a constant time, the overall time needed for computing $\tilde{\mathbf{d}}$ again becoming $O(m)$.

Thus, provided that we have all the necessary caches available, evaluating each feature requires $O(m)$ time, and hence one pass through the whole set of n features needs $O(mn)$ floating point operations. But we still have to ensure that the caches can be initialized and updated efficiently enough.

In the initialization phase of the greedy RLS algorithm (lines 1-4 in Algorithm 3) the set of selected features is empty, and hence the values of $\mathbf{a}$, $\mathbf{d}$, and C are initialized to $\lambda^{-1}\mathbf{y}$, $\lambda^{-1}\mathbf{1}$, and $\lambda^{-1}X^{\mathrm{T}}$, respectively, where $\mathbf{1} \in \mathbb{R}^m$ is a vector having every entry equal to 1. The computational complexity of the initialization phase is dominated by the $O(mn)$ time required for initializing C. Thus, the initialization phase is no more complex than one pass through the features.

When a new feature is added into the set of selected features, the vector $\mathbf{a}$ is updated according to (15) and the vector $\mathbf{d}$ according to (17). Putting together (13), (14), and (16), the cache matrix C can be updated via

$$C - \mathbf{u}(\mathbf{v}^{\mathrm{T}}C),$$

which requires $O(mn)$ time. This is equally expensive as the above introduced fast approach for trying each feature at a time using LOO as a selection criterion.

Finally, if we are to select altogether k features, the overall time complexity of greedy RLS becomes $O(kmn)$. The space complexity is dominated by the matrices X and C which both require $O(mn)$ space.

Algorithm 3: Greedy RLS algorithm proposed by us

Input: $X \in \mathbb{R}^{n\times m}$, $\mathbf{y} \in \mathbb{R}^m$, k, λ
Output: $\mathcal{S}$, $\mathbf{w}$

1 $\mathbf{a} \leftarrow \lambda^{-1}\mathbf{y}$;
2 $\mathbf{d} \leftarrow \lambda^{-1}\mathbf{1}$;
3 $C \leftarrow \lambda^{-1}X^{\mathrm{T}}$;
4 $\mathcal{S} \leftarrow \emptyset$;
5 **while** $|\mathcal{S}| < k$ **do**
6 $e \leftarrow \infty$;
7 $b \leftarrow 0$;
8 **foreach** $i \in \{1,\ldots,n\} \setminus \mathcal{S}$ **do**
9 $\mathbf{v} \leftarrow (X_i)^{\mathrm{T}}$;
10 $\mathbf{u} \leftarrow C_{:,i}(1 + \mathbf{v}C_{:,i})^{-1}$;
11 $\tilde{\mathbf{a}} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a})$;
12 $e_i \leftarrow 0$;
13 **foreach** $j \in \{1,\ldots,m\}$ **do**
14 $\tilde{\mathbf{d}}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j C_{j,i}$;
15 $p \leftarrow \mathbf{y}_j - (\tilde{\mathbf{d}}_j)^{-1}\tilde{\mathbf{a}}_j$;
16 $e_i \leftarrow e_i + l(\mathbf{y}_j, p)$;
17 **end**
18 **if** $e_i < e$ **then**
19 $e \leftarrow e_i$;
20 $b \leftarrow i$;
21 **end**
22 **end**
23 $\mathbf{v} \leftarrow (X_b)^{\mathrm{T}}$;
24 $\mathbf{u} \leftarrow C_{:,b}(1 + \mathbf{v}^{\mathrm{T}}C_{:,b})^{-1}$;
25 $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a})$;
26 **foreach** $j \in \{1,\ldots,m\}$ **do**
27 $\mathbf{d}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j C_{j,b}$;
28 **end**
29 $C \leftarrow C - \mathbf{u}(\mathbf{v}^{\mathrm{T}}C)$;
30 $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
31 **end**
32 $\mathbf{w} \leftarrow X_{\mathcal{S}}\mathbf{a}$;

4 Experimental results

In Section 4.1, we explore the computational efficiency of the introduced method and compare it with the best previously proposed implementation. In Section 4.2, we explore the quality of the feature selection process. In addition, we conduct measurements

of the degree to which the LOO criterion overfits in Section 4.3.

4.1 Computational efficiency

First, we present experimental results about the scalability of our method. We use randomly generated data from two normal distributions with 1000 features of which 50 are selected. The number of examples is varied. In the first experiment we vary the number of training examples between 500 and 5000, and provide a comparison to the low-rank updated LS-SVM method [19]. In the second experiment the training set size is varied between 1000 and 50000. We do not consider the baseline method in the second experiment, as it does not scale up to the considered training set sizes. The runtime experiments were run on a modern desktop computer with 2.4 GHz Intel Core 2 Duo E6600 processor, 8 GB of main memory, and 64-bit Ubuntu Linux 9.10 operating system.

We note that the running times of these two methods are not affected by the choice of the regularization parameter, or the distribution of the features or the class labels. This is in contrast to iterative optimization techniques commonly used to train for example support vector machines [26]. Thus we can draw general conclusions about the scalability of the methods from experiments with a fixed value for the regularization parameter, and synthetic data.

In Fig. 1 are the runtime comparisons with linear scaling on y-axis, and in Fig. 2 with logarithmic scaling. The results are consistent with the algorithmic complexity analysis of the methods. The method of [19] shows quadratic scaling with respect to the number of training examples, while the proposed method scales linearly. In Fig. 3 are the running times for the proposed method for up to 50000 training examples, in which case selecting 50 features out of 1000 took a bit less than twelve minutes.

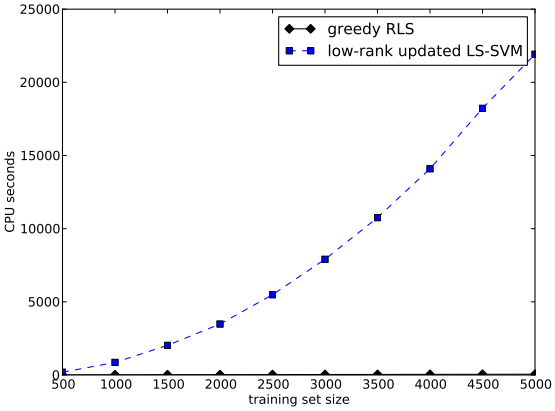

Figure 1: Running times in CPU seconds for the proposed greedy RLS method and the and the low-rank updated LS-SVM of [19]. Linear scaling on y-axis.

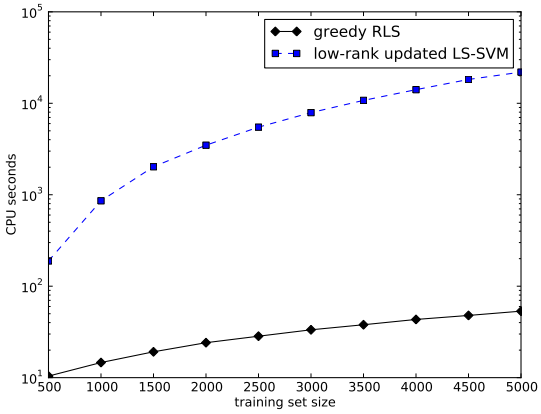

Figure 2: Running times in CPU seconds for the proposed greedy RLS method and the and the low-rank updated LS-SVM of [19] Logarithmic scaling on y-axis.

.

4.2 Quality of selected features

In this section, we explore the quality of the feature selection process. We recall that our greedy RLS algorithm leads to equivalent results as the algorithms

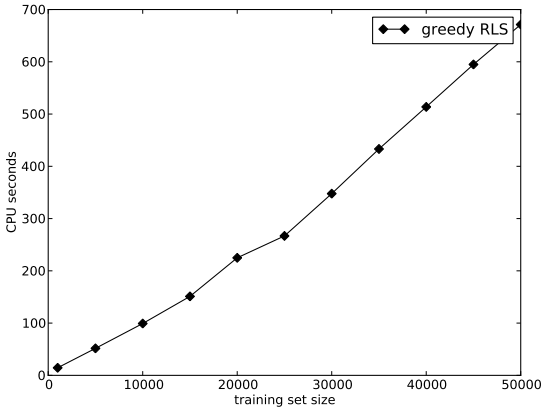

Figure 3: Running times in CPU seconds for the proposed greedy RLS method.

proposed by [17] and by [19], while being computationally much more efficient. The aforementioned authors have in their work shown that the approach compares favorably to other commonly used feature selection techniques. However, due to computational constraints the earlier evaluations have been run on small data sets consisting only of tens or hundreds of examples. We extend the evaluation to large scale learning with the largest of the considered data sets consisting of more than hundred thousand training examples. Wrapper selection methods do not typically scale to such data set sizes, especially when using cross-validation as a selection criterion. Thus we consider as a baseline an approach which chooses k features at random. This is a good sanity-check, since training RLS with this approach requires only $O(\min(k^2m, km^2))$ time that is even less than the time required by greedy RLS.

We consider the binary classification setting, the quality of a selected feature set is measured by the classification accuracy of a model trained on these features, evaluated on independent test data. The experiments are run on a number of publicly available benchmark data sets, whose characteristics are summarized in Table 1. The mnist dataset is originally a multi-class digit recognition task, here we consider the binary classification task of recognizing the digit 5.

Table 1: Data sets

data set	#instances	#features
adult	32561	123
australian	683	14
colon-cancer	62	2000
german.numer	1000	24
icjnn1	141691	22
mnist5	70000	780

All the presented results are average accuracies over stratified tenfold-cross validation run on the full data sets. On each of the ten cross-validation rounds, before the feature selection experiment is run we select the value of the regularization parameter as follows. We train the learner on the training folds using the full feature set, and perform a grid search to choose a suitable regularization parameter value based on leave-one-out performance.

Using the chosen regularization parameter value, we begin the incremental feature selection process on the training folds. One at a time we select the feature, whose addition to the model provides highest LOO accuracy estimate on the training folds. Each time a feature has been selected, the generalization performance of the updated model is evaluated on the test fold. The process is carried on until all the features have been selected.

In the presented figures we plot the number of selected features against the accuracy achieved on the test fold, averaged over the ten cross-validation rounds. In Fig. 4 are the results for the adult, in Fig. 5 for the australian, in Fig. 6 for colon-cancer, in Fig. 7 for german.numer, in Fig. 8 for ijcnn1, and in Figure 9 for the mnist5 data set.

On all data sets, the proposed approach clearly outperforms the random selection strategy, suggesting that the leave-one-out criterion leads to selecting informative features. In most cases with only a small subset of features one can achieve as good performance as with all the features.

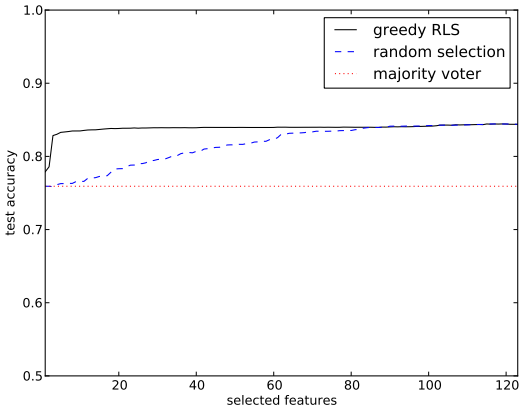

Figure 4: Performance on the adult data set.

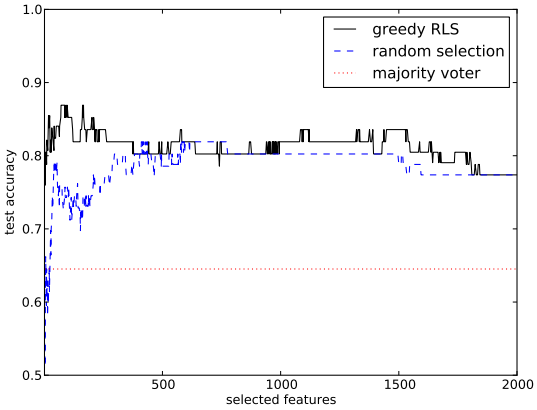

Figure 6: Performance on the colon-cancer data set.

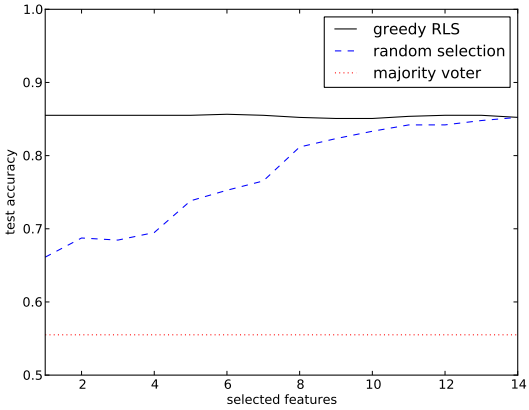

Figure 5: Performance on the australian data set.

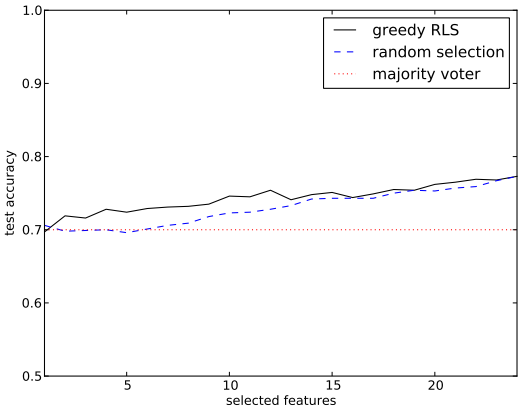

Figure 7: Performance on the german.numer data set.

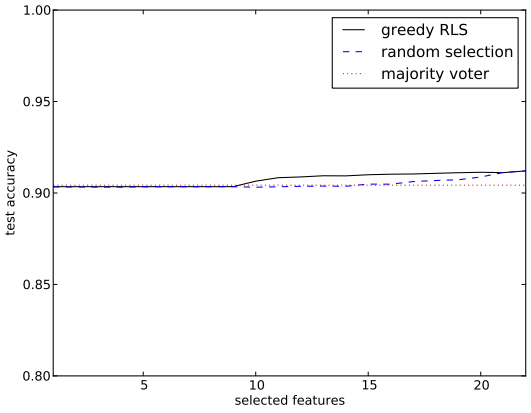

Figure 8: Performance on the ijcnn1 data set.

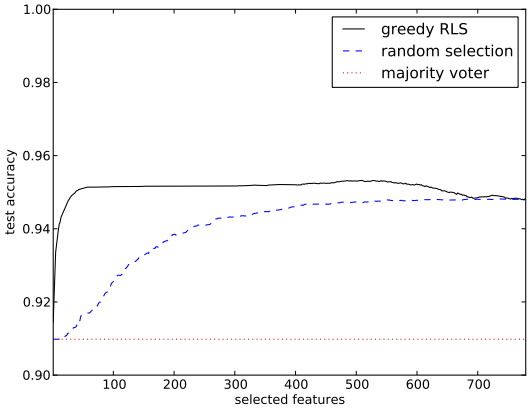

Figure 9: Performance on the mnist5 data set.

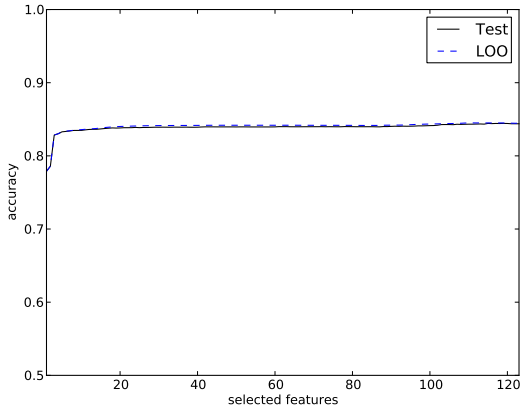

Figure 10: Test vs. LOO accuracy on the adult data set.

4.3 Overfitting in feature selection

One concern with using the LOO estimate for selecting features is that the estimate may overfit to the training data. If one considers large enough set of features, it is quite likely that some features will by random chance lead to improved LOO estimate, while not generalizing outside the training set. We explore to what extent and in which settings this is a problem by comparing the LOO and test accuracies in the previously introduced experimental setting. Again, we plot the number of selected features against the accuracy estimates.

In most cases the LOO and test accuracies are close to identical (Figures 10, 11, 14, and 15). However, on the colon-cancer and german.numer data sets (Figures 12 and 13) we see evidence of overfitting, with LOO providing over-optimistic evaluation of performance. The effect is especially clear on the colon-cancer data set, which has 2000 features, and only 62 examples. The results suggest that reliable feature selection can be problematic on small high-dimensional data sets, the type of which are often considered for example in bioinformatics. On larger data sets the LOO estimate is quite reliable.

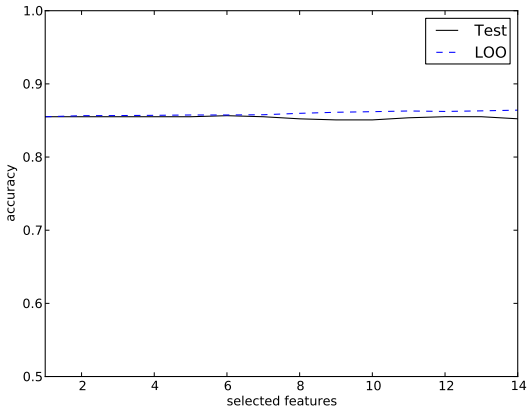

Figure 11: Test vs. LOO accuracy on the australian data set.

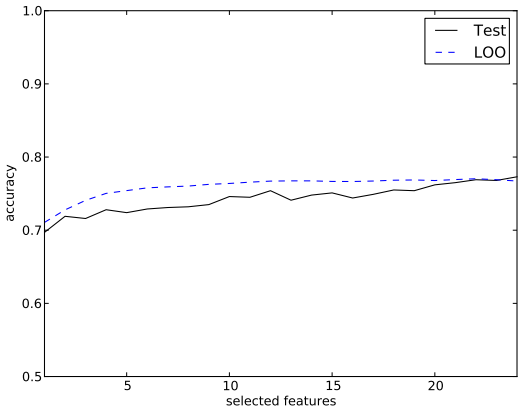

Figure 13: Test vs. LOO accuracy on the german.numer data set.

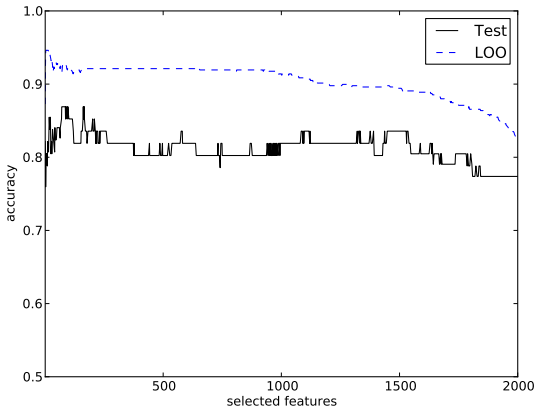

Figure 12: Test vs. LOO accuracy on the colon-cancer data set.

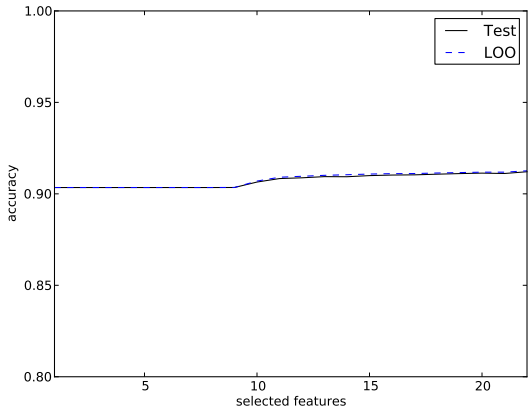

Figure 14: Test vs. LOO accuracy on the ijcnn1 data set.

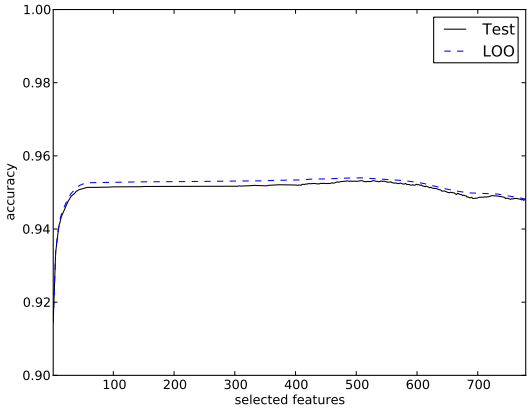

Figure 15: Test vs. LOO accuracy on the mnist5 data set.

5 Discussion and Future Directions

In this paper, we present greedy RLS, a linear time algorithm for feature selection for RLS. The algorithm uses LOO as a criterion for evaluating the goodness of the feature subsets and greedy forward selection as a search strategy in the set of possible feature sets. The proposed algorithm opens several directions for future research. In this section, we will briefly discuss some of the directions which we consider to be the most promising ones.

Greedy RLS can quite straightforwardly be generalized to use different types of cross-validation criteria, such as n-fold or repeated n-fold. These are motivated by their smaller variance compared to the leave-one-out and they have also been shown to have better asymptotic convergence properties for feature subset selection for ordinary least-squares [15]. This generalization can be achieved by using the short-cut methods developed by us [27] and by [28].

In this paper, we focus on the greedy forward selection approach but an analogous method can also be constructed for backward elimination. However, backward elimination would be computationally much more expensive than forward selection, because an RLS predictor has to be trained first with the whole feature set. In the feature selection literature, approaches which combine both forward and backward steps, such as the floating search methods (see e.g. [29, 30]), have also been proposed. Recently, [31] considered a modification of the forward selection for least-squares, which performs corrective steps instead of greedily adding a new feature in each iteration. The considered learning algorithm was ordinary least-squares and the feature subset selection criterion was empirical risk. The modified approach was shown to have approximately the same computational complexity (in number of iterations) but better performance than greedy forward selection or backward elimination. A rigorous theoretical justification was also presented for the modification. While this paper concentrates on the algorithmic implementation of the feature subset selection for RLS, we aim to investigate the performance of this type of modifications in our future work.

The computational short-cuts presented in this paper are possible due to the closed form solution the RLS learning algorithms has. Such short-cuts are also available for variations of RLS such as RankRLS, an RLS based algorithm for learning to rank proposed by us in [32, 33]. In our future work, we also plan to design and implement similar feature selection algorithms for RankRLS.

Analogously to the feature selection methods, many approaches has been developed also for so-called reduced set selection used in context of kernel-based learning algorithms (see e.g [10, 34] and references therein). For example, incremental approaches for selecting this set for regularized least-squares has been developed by [35]. Closely related to the reduced set methods, we can also mention the methods traditionally used for selecting centers for radial basis function networks. Namely, the algorithms based on the orthogonal least-squares method proposed by [36] for which efficient forward selection methods are also known. In the future, we plan to investigate how well approaches similar to our feature selection algorithm could perform on the tasks of reduced set or center selection.

6 Conclusion

We propose greedy regularized least-squares, a novel training algorithm for sparse linear predictors. The predictors learned by the algorithm are equivalent with those obtained by performing a greedy forward feature selection with leave-one-out (LOO) criterion for regularized least-squares (RLS), also known as the least-squares support vector machine or ridge regression. That is, the algorithm works like a wrapper type of feature selection method which starts from the empty feature set, and on each iteration adds the feature whose addition provides the best LOO performance. Training a predictor with greedy RLS requires $O(kmn)$ time, where k is the number of non-zero entries in the predictor, m is the number of training examples, and n is the original number of features in the data. This is in contrast to the computational complexity $O(\min\{k^3m^2n, k^2m^3n\})$ of using the standard wrapper method with LOO selection criterion in case RLS is used as a black-box method, and the complexity $O(km^2n)$ of the method proposed by [19], which is the most efficient of the previously proposed speed-ups. Hereby, greedy RLS is computationally more efficient than the previously known feature selection methods for RLS.

We demonstrate experimentally the computational efficiency of greedy RLS compared to the best previously proposed implementation. In addition, we explore the quality of the feature selection process and measure the degree to which the LOO criterion overfits with different sizes of data sets.

We have made freely available a software package called RLScore out of our previously proposed RLS based machine learning algorithms[2]. An implementation of the greedy RLS algorithm will also be made available as a part of this software.

References

[1] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[2] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[3] S. S. Keerthi and S. K. Shevade, "A fast tracking algorithm for generalized lars/lasso," *IEEE Transactions on Neural Networks*, vol. 18, no. 6, pp. 1826–1830, 2007.

[4] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of the Eleventh International Conference on Machine Learning*, W. W. Cohen and H. Hirsch, Eds. San Fransisco, CA: Morgan Kaufmann Publishers, 1994, pp. 121–129.

[5] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.

[6] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, 1990.

[7] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," in *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 515–521.

[8] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[9] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific Pub. Co., Singapore, 2002.

[10] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," in *Advances in Learning Theory: Methods, Model and Applications*, ser. NATO Science Series III: Computer and System Sciences, J. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, Eds.

[2] Available at http://www.tucs.fi/RLScore

Amsterdam: IOS Press, 2003, vol. 190, ch. 7, pp. 131–154.

[11] T. Poggio and S. Smale, "The mathematics of learning: Dealing with data," *Notices of the American Mathematical Society (AMS)*, vol. 50, no. 5, pp. 537–544, 2003.

[12] V. Vapnik, *Estimation of Dependences Based on Empirical Data [in Russian]*. Moscow, Soviet Union: Nauka, 1979, (English translation: Springer, New York, 1982).

[13] G. Wahba, *Spline Models for Observational Data*. Philadelphia, USA: Series in Applied Mathematics, Vol. 59, SIAM, 1990.

[14] P. Green and B. Silverman, *Nonparametric Regression and Generalized Linear Models, A Roughness Penalty Approach*. London, UK: Chapman and Hall, 1994.

[15] J. Shao, "Linear model selection by cross-validation," *Journal of the American Statistical Association*, vol. 88, no. 422, pp. 486–494, 1993.

[16] P. Zhang, "Model selection via multifold cross validation," *The Annals of Statistics*, vol. 21, no. 1, pp. 299–313, 1993.

[17] E. K. Tang, P. N. Suganthan, and X. Yao, "Gene selection algorithms for microarray data based on least squares support vector machine," *BMC Bioinformatics*, vol. 7, p. 95, 2006.

[18] Z. Ying and K. C. Keong, "Fast leave-one-out evaluation for dynamic gene selection," *Soft Computing*, vol. 10, no. 4, pp. 346–350, 2006.

[19] F. Ojeda, J. A. Suykens, and B. D. Moor, "Low rank updated LS-SVM classifiers for fast variable selection," *Neural Networks*, vol. 21, no. 2-3, pp. 437 – 449, 2008, advances in Neural Networks Research: IJCNN '07.

[20] S. R. Searle, *Matrix Algebra Useful for Statistics*. New York, NY: John Wiley and Sons, Inc., 1982.

[21] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, pp. 181–201, 2001.

[22] B. Schölkopf and A. J. Smola, *Learning with kernels*. MIT Press, Cambridge, MA, 2002.

[23] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge: Cambridge University Press, 2004.

[24] R. Rifkin and R. Lippert, "Notes on regularized least squares," Massachusetts Institute of Technology, Tech. Rep. MIT-CSAIL-TR-2007-025, 2007.

[25] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[26] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large-Scale Kernel Machines*, ser. Neural Information Processing, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 1–28.

[27] T. Pahikkala, J. Boberg, and T. Salakoski, "Fast n-fold cross-validation for regularized least-squares," in *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, T. Honkela, T. Raiko, J. Kortela, and H. Valpola, Eds. Espoo, Finland: Otamedia, 2006, pp. 83–90.

[28] S. An, W. Liu, and S. Venkatesh, "Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression," *Pattern Recognition*, vol. 40, no. 8, pp. 2154–2162, 2007.

[29] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern Recogn. Lett.*, vol. 15, no. 11, pp. 1119–1125, 1994.

[30] J. Li, M. T. Manry, P. L. Narasimha, and C. Yu, "Feature selection using a piecewise linear network," *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1101–1115, 2006.

[31] T. Zhang, "Adaptive forward-backward greedy algorithm for sparse learning with linear models," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, pp. 1921–1928.

[32] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and T. Salakoski, "Learning to rank with pairwise regularized least-squares," in *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, Eds., 2007, pp. 27–33.

[33] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and J. Järvinen, "An efficient algorithm for learning to rank from preference graphs," *Machine Learning*, vol. 75, no. 1, pp. 129–165, 2009.

[34] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, P. Langley, Ed. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2000, pp. 911–918.

[35] L. Jiao, L. Bo, and L. Wang, "Fast sparse approximation for least squares support vector machine," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 685–697, 2007.

[36] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, 1991.